Evolutionary design of graph-based structures for optical computing

Mihai Oltean, Oana Muntean

Department of Computer Science,
Faculty of Mathematics and Computer Science,
Babeş-Bolyai University, Kogălniceanu 1,
Cluj-Napoca, 400084, Romania.
mihai.oltean@gmail.com
oana_muntean85@yahoo.com
https://mihaioltean.github.io/optical

Abstract. Designing optical devices for solving NP-complete problems is a difficult task. The difficulty consists in constructing a graph which - when traversed by light - generates all possible solutions of the problem to be solved. So far only few devices of this type have been proposed. Here we suggest the use of evolutionary algorithms for solving this problem: the graphs are generated using a special Genetic Programming approach. We have tested our idea on the subset sum problem. Numerical experiments shows the effectiveness of the proposed approach.

keywords: evolutionary algorithms, genetic programming, optical computing, unconventional computing, NP-complete

1 Introduction

The purpose of this reseach is to asses the usefulness of Evolutionary Algorithms (EAs) [8, 12, 20] for designing graph-based devices for delay-based optical computing systems.

A common feature of all these devices is the fact that the signals are delayed by a certain amount of time. The existence of a solution is determined by checking whether there is at least one signal which was delayed by a precise amount of time. If we don't find a signal at that moment it means that the problem has no solution.

The difficulty of this approach resides in the design of a delaying system such that the solution can simply be read at an exact moment of time [25]. Up to now these graphs have been manually designed for only few problems: Hamiltonian path [5, 6, 9, 10, 21, 22], Travelling Salesman problem [5, 6, 9, 10], subset sum [6, 23], Diophantine equations [16, 18], Exact Cover [24], Clique [5, 6], Vertex Cover [5, 6], 3-Sat [5, 6], 3D-matching [5, 6], the unbounded subset sum [17] and Ricochet Robot [11].

Here we use a Genetic Programming [12] variant called Multi Expression Programming (MEP) [20, 26] for automatic generation of graphs fitting the requirements imposed by the optical computing. Each MEP chromosome encodes a possible solution for the problem. Each gene of the chromosome represents a node of the graph together with its neighbors and associated delays. When the evaluation of individuals takes place, all possible paths for the signal are generated.

Offspring obtained by crossover and mutation are always syntactically correct MEP individuals. Thus, no extra processing for repairing newly obtained individuals is needed.

We exemplify our approach by evolutionary designing the graph for the subset sum problem [7, 23]. For this purpose we have performed several numerical experiments with various settings for our evolutionary algorithm. Instances up to 4 numbers in the set have been successfully tackled with reasonable populations and number of generations.

The paper is organized as follows: Section 2 describes some basic principles of delayed based optical devices. Section 3 contains a short description of the subset sum problem. The graph where the light is running to form a solution is described in section 4. Section 5 contains a brief description of Evolutionary Algorithms and Genetic Programming. Next section (6) gives a short description of standard Multi Expression Programming. Section 7 deeply describes the proposed evolutionary system. It contains details about the representation, initialization, genetic operators and fitness function. Several numerical experiments are performed in section 8. Section 9 concludes our paper.

2 The manually designed optical devices

This section describes some elements behind the delay-based optical devices used for solving NP-complete problems. For more information the reader is asked to read the following papers: [23, 25].

The idea is based on two properties of light (signal):

- The speed of light has a limit. We can delay the ray by forcing it to pass through an optical fiber cable of a certain length.
- The ray can be easily divided into multiple rays of smaller intensity/power. Beam-splitters are used for this operation.

The optical devices have a graph like structure. Generally speaking one operation is performed when a ray passes through a node and one operation is performed when a ray passes through an edge.

- When passing through an arc the light ray is delayed by the amount of time assigned to that arc.
- When the ray is passing through a node it is divided into a number of rays equal to the external degree of that node. Each obtained ray is directed toward one of the nodes connected to the current node.

3 The subset sum problem

We exemplify our idea with a classic NP-complete problem: the subset sum [7] which can be simply stated as:

Given a set of positive numbers $A = \{a_1, a_2, ..., a_n\}$ and another positive number B. Is there a subset of A whose sum equals B?

We focus our attention on the YES / NO decision problem. We are not interested in finding the subset generating the solution. Actually we are interested to find only if such subset does exist.

4 The graph for the subset sum

The most important part of the device is the graph which - when traversed by light - generates all possible solutions for the problem.

In the current case that we analyze (the subset sum problem) numbers from the given set A represent the delays induced to the signals (light) that passes through the device. For instance, if numbers a_1, a_3 and a_7 generate the expected subset, then the total delay of the signal should be $a_1 + a_3 + a_7$.

Thus our graph is composed of a set of nodes and a set of arcs whose length are taken from the given set A. We also need arcs of 0 lengths whose purpose is to avoid the use of a given number from the initial set.

Such device [23] is depicted in Figure 1. A light ray sent to start node will have the possibility to either traverse a given arc (from the upper part of figure) or to skip it (by traversing the arc of length 0 from the bottom of figure).

In each node (but the last one) a beam-splitter is placed which will split a ray into 2 subrays of smaller intensity.

The device will generate all possible subsets of A. Each subset will delay one of the rays by an amount of time equal to the sum of the lengths of the arcs in that path.

In the graph depicted in Figure 1 the light will enter in *Start* node. It will be divided into 2 subrays of smaller intensity. These 2 rays will arrive into the second node at moments a_1 and 0. Each of them will be divided into 2 subrays which will arrive in the 3^{rd} node at moments $0, a_1, a_2, a_1 + a_2$. These rays will arrive at no more than 4 different moments.

In the destination node the rays arrive at no more than 2^n different moments. The ray arriving at moment 0 means the empty set. The ray arriving at moment $a_1 + a_2 + ... + a_n$ represents the full set. If there is a ray arriving at moment B means that there is a subset of A of sum B.

5 Evolutionary Algorithms and Genetic Programming

Evolutionary Algorithms (EAs) [8] are approximation tools for solving difficult real-world problems. They were developed under the pressure generated by the inability of classical (mathematical) methods to solve some real-world problems.

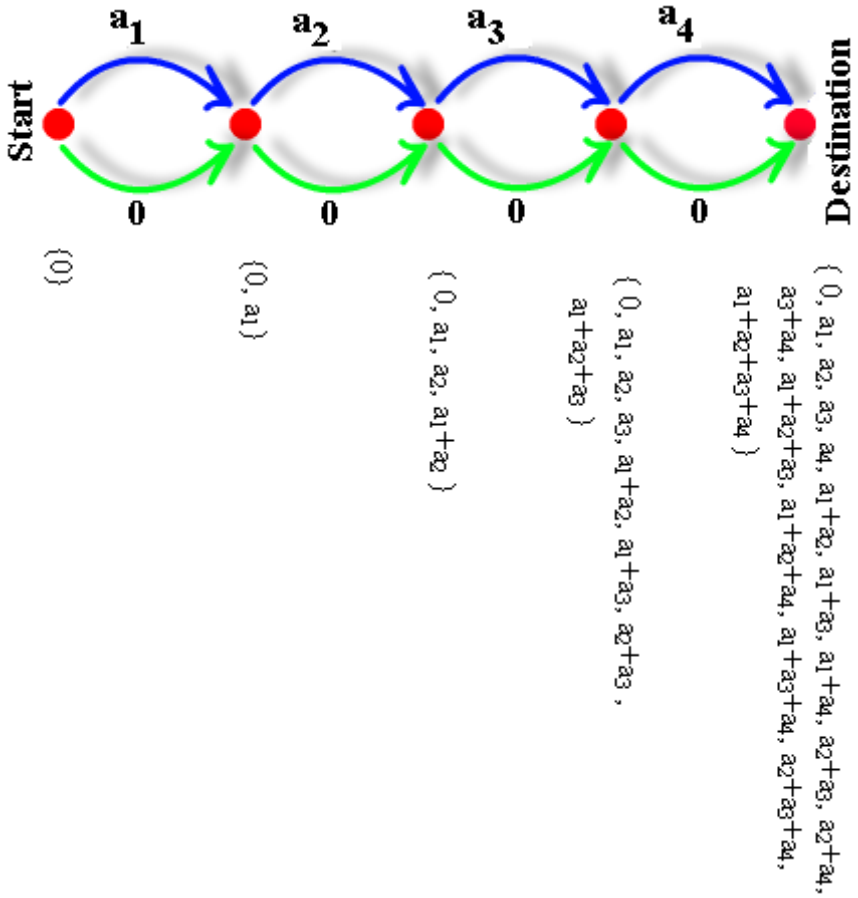

Fig. 1. The device for the subset sum problem. Each subset of A is generated. Skipping arcs have 0 lengths. We have also depicted the moments when different rays arrive in nodes. The moments are represented as sets because they might not be distinct

Many of these unsolved problems are (or could be turned into) optimization problems. Solving an optimization problem means finding of solutions that maximize or minimize a criteria function [20].

One of the most important algorithms belonging to this class is Genetic Programming (GP) [12–14]. Genetic Programming is widely known as the technique which writes computer programs. Instead of evolving solutions for a particular problem instance, GP is mainly intended for discovering computer programs able to solve classes of problems.

Many variants of GP have been proposed in the recent years. Their aims were various: simpler implementation, higher speed, smaller memory requirements, the capability of working with particular hardware architectures etc. Another motivation is given by the problems where some representations work better than the others [26].

One of the GP variants is Multi Expression Programming (MEP) [20, 26]. This technique will be used in this paper for performing several numerical experiments. In the next section we briefly describe standard Multi Expression Programming and then we modify it for adapting it to our current purpose.

6 Multi Expression Programming

Multi Expression Programming (MEP) [20] is a Genetic Programming variant that uses a linear representation of chromosomes. MEP individuals are strings of genes encoding complex computer programs.

When MEP individuals encode expressions, their representation is similar to the way in which compilers translate C or *Pascal* expressions into machine code.

An example of chromosome C using the function set $F = \{+, *\}$ and the set of terminals $T = \{a, b, c, d\}$ is given below:

1: a
2: b
3: + 1, 2
4: c
5: d
6: + 4, 5
7: * 3, 5
8: + 2, 6

A unique MEP feature is the ability of storing multiple solutions of a problem in a single chromosome. What we have encoded in chromosome C are the following expressions:

$E_1 = a$
$E_2 = b$
$E_4 = c$
$E_5 = d$
$E_3 = a + b$

$E_6 = c + d$
$E_7 = (a + b) * d$
$E_8 = b * (c + d)$

Usually, the best solution is chosen for fitness assignment. When solving symbolic regression or classification problems (or any other problems for which the training set is known before the problem is solved) MEP has the same complexity as other techniques storing a single solution in a chromosome.

Evaluation of the expressions encoded into a MEP individual can be performed by a single parsing of the chromosome.

Offspring obtained by crossover and mutation are always syntactically correct MEP individuals (computer programs). Thus, no extra processing for repairing newly obtained individuals is needed.

There are 2 main differences between GP and MEP. First of all MEP encodes multiple solutions instead of one and secondly, MEP uses a linear representation while GP has a tree based representation of solutions. Linear encoding of computer programs means that we usually work with arrays of fixed or variable lengths. Specifically:

1. we generate arrays of instructions, having a particular meaning,
2. we recombine them by using string-based crossover operators such as those from binary encoding (such as one-cutting point, two cutting points, uniform recombination etc.) [8],
3. we mutate them using operators inspired from the binary encoding or from other representations [8].

7 The proposed MEP-based approach

Here we deeply describe the proposed evolutionary approach. We have made several modifications to standard MEP in order to make it suitable for our purpose.

7.1 Representation

MEP genes are strings of a variable length. The number of genes per chromosome is fixed. This number defines the length of the chromosome. Even if this number if fixed we still can have solutions of variable length because usually not all genes are utilized.

Each gene represent a node and stores the arcs leaving that node (the adjacency list) and the length for those arcs.

Cycles can appear in this structure because there is no restriction on why what kind of nodes appear in the adjacency list. This makes the fitness function computationally expensive and difficult to compute.

There is still one natural restriction: a node cannot appear multiple times in the adjacency list of another node. A node may appear in its own adjacency list.

Example

Consider a representation where the numbers on the left positions stand for gene labels. Labels do not belong to the chromosome. They represent the node index.

Since all nodes are identical (contain a beam splitter) we don't need a function set as in the standard MEP.

The set of terminals is made of a set of nodes and the possible lengths for the arcs connecting the current node with node from its adjacency list.

Suppose that we deal with a problem with a set of 4 numbers: $A = \{a_1, a_2, a_3, a_4\}$. In this case the terminal set could be:

$T = \{1, 2, 3, ..., a_1, a_2, a_3, a_4, 0\}$, where $1, 2, 3, ...$ are the nodes and $a_1, a_2, a_3, a_4, 0$ are the arcs' length. The number of nodes employed by our solution is not known the beginning. The maximal number of nodes is equal the chromosome length. We do provide a large initial chromosome length.

The length of each arc is given right after a node.

Choosing the length for each arc is a difficult task. For some problems (such as the subset sum) the lengths are taken directly from the problem's input, but for some other cases (take for instance the Hamiltonian path [21]) the lengths have complicated formulas: $2^n - 2^i$, where $0 \leq i \leq n$. In this case we also need to use Genetic Programming for generating complex mathematical formulas. However, for the case currently analyzed we will simplify this aspect by using only some values taken from the problem input.

An example of chromosome C is given below:

1: 3, a_2, 5, 0
2: 4, a_1, 2, 0, 6, a_4
3: 6, 0
4: 1, a_2, 2, a_1, 5, a_3
5: 2, a_4
6: 4, a_4, 3, 0

This chromosome must be interpreted as follows:

- The maximal number of nodes in a solution is 6. However, any number between 1 and 6 can be the actual solution since not all nodes must be used.
- Node 1 has 2 out nodes: 3 and 5. The arc (1,3) has length a_2 and arc (1,5) has length 0.
- Node 2 has 3 out nodes: 4, 2 and 6. The arc (2,4) has length a_1, arc (2,2) has length 0 and arc (2,6) has length a_4. Note here that 2 has an arc to itself.
- Node 3 has 1 out node: 6. The arc (3,6) has length 0.
- Node 4 has 3 out nodes: 1, 2 and 5. The arc (4,1) has length a_2, arc (4,2) has length a_1 and arc (4,5) has length a_3.
- Node 5 has 1 out node: 2. The arc (5,2) has length a_4.
- Node 6 has 2 out nodes: 4 and 3. The arc (6,4) has length a_4 and arc (6,3) has length 0.

As explained in section 2 each device has a *start* node and a *destination* node. In our case the *start* node is node 1 and the *destination* node will be computed during the fitness evaluation.

7.2 Initialization

Generation of chromosomes is done randomly. For each node we randomly choose the number of nodes in its adjacency list and then we randomly generate each node in the list. After each node we randomly choose the length of the arc connecting that node with its predecesor.

7.3 Fitness assignment

During fitness evaluation we compute how good the current chromosome is.

For the subset sum problem which is investigated here we want to generate all possible subsets of the given set.

A perfect solution is that one which contains a destination node in which arrive delayed rays encoding all subsets of the given set. A less perfect solution is the one for which not all subsets are generated. This is why a natural way to define the fitness is to count how many subsets have been generated. 2^n is the best fitness possible while 0 is the worst fitness possible.

For computing the fitness we have no choice but to simulate the signal passing through the device. Of course, this operation requires an exponential amount of memory because the number of signals grows exponentially. Partial results are stored similarly with Dynamic Programming [3].

Also, for each signal we have to store its entire path. When the signal is divided by the beam-splitters each subsignal will inherit the information carried by its parent signal.

Because we want to limit the amount of computer resources involved in this operation we have imposed the following constrains:

- When the total delay of a given signal exceeds a certain threshold we stop propagating that signal. In the case of the subset sum problem we stop when the delay of a signal is larger than B. It makes no sense to continue since we are interested only in finding a subset of sum B.
- Because we allow arcs of length 0, we can have a huge number of divisions of the signal without increasing the total delay of that signal. In this case we have imposed a limit on the total number of divisions that a signal can have.

The problem now is which node will be the destination node. There is neither practical nor theoretical evidence that one of the nodes is better than the others. Moreover, Wolpert and McReady [30] with their well known theorems for No Free Lunch for Search and Optimization proved that we cannot use the search algorithm's behavior so far for a particular test function to predict its future behavior on that function.

Preserving the MEP basic-idea with multiple solutions in the same chromosome we say again that the fitness of the entire chromosome is set as the fitness of the best solution encoded by that chromosome. In our case the fitness is given by the best *destination* node (in which arrive the highest number of signals encoding subsets of A) in that device. There is no extra cost of doing that because when the signal was propagated through the graph it is propagated through all reachable nodes.

Note that if some other signals not encoding subsets arrive in a node, the quality of that node is decreased correspondingly. Thus, we can have negative fitnesses too.

In is obvious that some parts of a MEP chromosome are not used. Some GP techniques, like Linear GP, remove non-coding sequences of chromosome during the search process. As already noted [4] this strategy does not give the best results. The reason is that sometimes, a part of the useless genetic material has to be kept in the chromosome in order to maintain population diversity.

Other techniques such as Cartesian GP (CGP) [15] employ a different strategy for selecting the nodes which provide the output: they simply evolve these nodes as part of the chromosome. This is also good, but during numerical experiments we have noticed that selecting the best solution found works better than evolving it.

7.4 Search operators

The search operators used within MEP algorithm are crossover and mutation. These search operators preserve the chromosome structure.

Crossover By crossover two parents are selected and are recombined.

Two variants of recombination have been considered and tested within our MEP implementation: one-point recombination and uniform recombination.

One-point recombination

One-point recombination operator in MEP representation is similar to the corresponding binary representation operator. One crossover point is randomly chosen and the parent chromosomes exchange the sequences at the right side of the crossover point.

Example

Consider the parents C_1 and C_2 given below. Choosing the crossover point after position 3 two offspring, O_1 and O_2 are obtained as given in Table 1.

Uniform recombination

During the process of uniform recombination, offspring genes are taken randomly from one parent or another.

Table 1. MEP one-point recombination.

Parents		Offspring	
C_1	C_2	O_1	O_2
1: 3, a_2, 5, 0	1: 2, 0	**1: 3, a_2, 5, 0**	1: 2, 0
2: 4, a_1, 2, 0, 6, a_4	2: 3, a_2, 1, 0	**2: 4, a_1, 2, 0, 6, a_4**	2: 3, a_2, 1, 0
3: 6, 0	3: 6, a_1	**3: 6, 0**	3: 6, a_1
4: 1, a_2, 2, a_1, 5, a_3	4: 4, a_2, 2, a_1, 1, a_2	4: 4, a_2, 2, a_1, 1, a_2	**4: 1, a_2, 2, a_1, 5, a_3**
5: 2, a_4	5: 5, a_4	5: 5, a_4	**5: 2, a_4**
6: 4, a_4, 3, 0	6: 1, a_3	6: 1, a_3	**6: 4, a_4, 3, 0**

Example

Let us consider the two parents C_1 and C_2 given below. The two offspring O_1 and O_2 are obtained by uniform recombination as given in Table 2.

Table 2. MEP uniform recombination.

Parents		Offspring	
C_1	C_2	O_1	O_2
1: 3, a_2, 5, 0	1: 2, 0	**1: 3, a_2, 5, 0**	1: 2, 0
2: 4, a_1, 2, 0, 6, a_4	2: 3, a_2, 1, 0	2: 3, a_2, 1, 0	**2: 4, a_1, 2, 0, 6, a_4**
3: 6, 0	3: 6, a_1	3: 6, a_1	**3: 6, 0**
4: 1, a_2, 2, a_1, 5, a_3	4: 4, a_2, 2, a_1, 1, a_2	4: 4, a_2, 2, a_1, 1, a_2	**4: 1, a_2, 2, a_1, 5, a_3**
5: 2, a_4	5: 5, a_4	**5: 2, a_4**	5: 5, a_4
6: 4, a_4, 3, 0	6: 1, a_3	6: 1, a_3	**6: 4, a_4, 3, 0**

Mutation Each symbol in the chromosome may be the target of the mutation operator. Some symbols in the chromosome are changed by mutation.

The number of nodes in the adjacency list of each node can be changed. Also the length associated to each arc can be changed.

Example

Consider the chromosome C given below. If the boldfaced symbols are selected for mutation an offspring O is obtained as given in Table 3.

Table 3. MEP mutation

C	O
1: 3, a_2, 5, 0	1: 3, a_2, 5, 0, **4, a_1**
2: 4, a_1, 2, 0, 6, a_4	2: 4, a_1, 2, **a_5**
3: 6, 0	3: 6, 0
4: 1, a_2, 2, a_1, 5, a_3	4: 1, a_2, 2, **a_3**, 5, a_3
5: 2, a_4	5: 2, a_4, **1, a_2**
6: 4, a_4, 3, 0	6: 4, a_4, 3, 0

7.5 MEP algorithm

Standard MEP algorithm uses steady-state evolutionary model [29] as its underlying mechanism.

The MEP algorithm starts by creating a random population of individuals. The following steps are repeated until a given number of generations is reached: Two parents are selected using a standard selection procedure. The parents are recombined in order to obtain two offspring. The offspring are considered for mutation. The best offspring O replaces the worst individual W in the current population if O is better than W.

The variation operators ensure that the chromosome length is a constant of the search process. The algorithm returns as its answer the best solution evolved along a fixed number of generations.

The standard MEP algorithm is outlined below:

Standard MEP Algorithm

S_1. Randomly create the initial population $P(0)$
S_2. **for** $t = 1$ **to** *Max_Generations* **do**
S_3. **for** $k = 1$ **to** $|P(t)|$ / 2 **do**
S_4. p_1 = *Select*($P(t)$); // select one individual from the current population
S_5. p_2 = *Select*($P(t)$); // select the second individual
S_6. *Crossover* (p_1, p_2, o_1, o_2); // crossover the parents p_1 and p_2
// the offspring o_1 and o_2 are obtained
S_7. *Mutation* (o_1); // mutate the offspring o_1
S_8. *Mutation* (o_2); // mutate the offspring o_2
S_9. **if** *Fitness*(o_1) > Fitness(o_2)
S_{10}. **then if** *Fitness*(o_1) > the fitness of the worst individual in the current population
S_{11}. **then** *Replace* the worst individual with o_1;
S_{12}. **else if** *Fitness*(o_2) > the fitness of the worst individual in the current population
S_{13}. **then** *Replace* the worst individual with o_2;

S_{14}. **endfor**
S_{15}. **endfor**

8 Numerical experiments

In this section we perform several numerical experiments for evolving graph structures for optical computing devices. Subset sum problem with sets of 3 and 4 numbers are used. For larger sets we can design a solution by generalizing from the smaller sets.

The parameters of the MEP algorithm are given in Table 4.

Table 4. The parameters for the MEP algorithm used for discovering solutions for the subset sum problem with 3 and 4 numbers.

Parameter	Value
Population size	500
Number of generations	50
Chromosome length	8
Maximal divisions allowed per signal	10
Mutation probability	0.2
Crossover type	Uniform
Crossover probability	0.9
Selection	Binary tournament
Number of runs	30

We have performed 30 independent runs with different seeds because evolutionary algorithms use random numbers and a single run is not conclusive.

For the subset sum problem with 3 nodes the success rate was 90%. That is in 27 out of 30 runs we have obtained a perfect solution (all 2^3 subsets have been generated in the *destination* node).

For the subset sum problem with 4 nodes the success rate was 36%. That is in 11 out of 30 runs we have obtained a perfect solution (all 2^4 subsets have been generated in the *destination* node). It is obvious that the difficulty of the problem will increase as the size of the set increases.

In almost all cases the standard design (see Figure 1) was obtained. However, in some particular runs, some other strange designs were obtained (which are not presented here due to the space limitation). We expected to obtain such designs because usually the evolution can follow a path which is not always the logical one. However, the strange designs were not as efficient (in the number of nodes and arcs) as the standard design. This is why we have not focused too much on analyzing them.

9 Conclusions and future work

Here we have used evolutionary algorithms as an automatic tool for designing graphs for optical delay-based systems. Numerical experiments have shown good results for several small instances of the subset sum problem. Large instances can be easily solved by generalizing the design obtained in the experiments.

Further work directions are focused on:

- Improving the speed of MEP by using Sub Machine Code GP [27, 28]. Experiments performed in [27] have shown that speed of GP was increased more than 1 order of magnitude. This will help us to solve larger instances of the problem.
- Improving the search by using Automatically Defined Functions (ADFs) [13]. This will help us to solve larger instances due to the generalization ability of ADFs.
- Using Genetic Programming for generating the length of arcs required by the device. Some other problems, such as the Hamiltonian path, use complex formulas for expressing the length based on the values given as input.
- Analyzing the relationship between the success rate and different parameters of the algorithm (such as the population size, number of generations, search operators probability etc). This could help us to solve larger instances of the problem.

Acknowledgment

This work was supported by grant CNCSIS-IDEI-543/2007.

The source code for Multi Expression Programming can be downloaded from `https://mepx.github.io`.

References

1. Aaronson, S.: NP-complete problems and physical reality. ACM SIGACT News Complexity Theory Column, March. ECCC TR05-026, quant-ph/0502072 (2005)
2. Banzhaf, W., Nordin, P., Keller, E. R., Francone, F. D.: Genetic Programming - An Introduction, Morgan Kaufmann, San Francisco, CA (1998)
3. Bellman, R.: Dynamic Programming, Princeton, Princeton University Press, New Jersey, (1957)
4. Brameier, M., Banzhaf, W.: A Comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining, IEEE Transactions on Evolutionary Computation, Vol. 5, pp. 17-26, IEEE Press, NY (2001)
5. Dolev, S. Nir, Y.: Optical Implementation of Bounded non Deterministic Turing Machine, Patent Filed May 2003 in Israel, May 2004 USA (2004)
6. Dolev, S., Fitoussi, H.: The Traveling Beams, Optical Solutions for Bounded NP-Complete Problems, Fourth International Conference on Fun with Algorithms (FUN2007),LNCS 4475, pp. 120-134 (2007)

7. Garey, MR., Johnson, DS.: Computers and intractability: A guide to NP-Completeness. Freeman & Co, San Francisco, CA (1979)
8. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, MA (1989)
9. Haist, T., Osten, W.: An Optical Solution For The Traveling Salesman Problem. Opt. Express, Vol. 15, 10473-10482 (2007)
10. Haist, T., Osten, W.: An Optical Solution For The Traveling Salesman Problem:erratum. Opt. Express, Vol. 15, 12627-12627 (2007)
11. Haist, T., Osten, W.: Ultra-fast digital optical arithmetic using waveoptical computing, OSC 2008, LNCS 5172, pp. 33-45 (2008)
12. Koza, J. R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA, (1992)
13. Koza, J. R.: Genetic Programming II: Automatic Discovery of Reusable Subprograms, MIT Press, Cambridge, MA (1994)
14. Koza, J. R. et al.: Genetic Programming III: Darwinian Invention and Problem Solving, Morgan Kaufmann, San Francisco, CA (1999)
15. Miller, J.F., Thomson, P.: Cartesian Genetic Programming. In Proceedings of the 3^{rd} International Conference on Genetic Programming (EuroGP2000), R. Poli, J.F. Miller, W. Banzhaf, W.B. Langdon, J.F. Miller, P. Nordin, T.C. Fogarty (Editors), LNCS 1802, Springer-Verlag, Berlin, pp. 15-17 (2000)
16. Muntean, O.: Optical Solutions for NP-complete problems, graduation thesis, Faculty of Mathematics and Computer Science, Babes-Bolyai University, Cluj-Napoca, Romania, defended 3^{rd} of July (2007)
17. Muntean, O., Oltean, M.: Using light for solving the unbounded subset-sum problem, International Journal of Innovative Computing, Information and Control, Vol. 5, Issue 8, pp. 2159-2167, (2009)
18. Muntean, O., Oltean, M.: Deciding whether a linear Diophantine equation has solutions by using a light-based device, (submitted) (2008)
19. Nordin, P.: A Compiling Genetic Programming System that Directly Manipulates the Machine Code, K. E. Kinnear, Jr. (editor), Advances in Genetic Programming, pp. 311-331, MIT Press (1994)
20. Oltean M, Groşan C.: Evolving Evolutionary Algorithms using Multi Expression Programming, The 7^{th} European Conference on Artificial Life, Dortmund, 14-17 September, Banzhaf W, (editor), LNAI 2801, pp. 651-658, Springer-Verlag, Berlin (2003)
21. Oltean, M.: A light-based device for solving the Hamiltonian path problem. Unconventional Computing, Calude C. (et al.) (Eds), LNCS 4135, Springer-Verlag, 217-227 (2006)
22. Oltean, M.: Solving the Hamiltonian path problem with a light-based computer, Natural Computing, Springer-Verlag, Vol. 7, Issue 1, pp. 57-70, (2008)
23. Oltean, M., Muntean, O.: Solving the subset-sum problem with a light-based device, Natural Computing, Springer-Verlag, Vol. 8, Issue 2, pp. 321-331 (2009)
24. Oltean, M., Muntean, O.: Exact Cover with light, New Generation Computing, Springer-Verlag, Vol. 26, Issue 4, pp. 329-346 (2008)
25. Oltean, M., Muntean, O.: Solving NP-Complete Problems with Delayed Signals: An Overview of Current Research Directions, in proceedings of the 1^{st} International Workshop on Optical SuperComputing, LNCS 5172, Springer-Verlag, pp. 115-128 (2008)
26. Oltean, M., Grosan, C., Diosan, L., Mihaila, C.: Genetic Programming with linear representation: a survey, International Journal on Artificial Intelligence Tools, World Scientific, Vol. 19, Issue 2, pp. 197-238 (2009)

27. Poli R., Langdon W B.: Sub-machine Code Genetic Programming, in Advances in Genetic Programming 3, L. Spector, W. B. Langdon, U.-M. O'Reilly, P. J. Angeline, Eds. Cambridge:MA, MIT Press, chapter 13, (1999)
28. Poli, R., Page, J.: Solving High-Order Boolean Parity Problems with Smooth Uniform Crossover, Sub-Machine Code GP and Demes, Journal of Genetic Programming and Evolvable Machines, Kluwer, pp. 1-21 (2000)
29. Syswerda, G.: Uniform Crossover in Genetic Algorithms, Proceedings of the 3^{rd} International Conference on Genetic Algorithms, Schaffer, J.D., (editor), MKP, CA, pp. 2-9 (1989)
30. Wolpert, D.H., McReady, W.G.: No Free Lunch Theorems for Optimisation, IEEE Transaction on Evolutionary Computation, Vol. 1, pp. 67-82 (1997)